jarjar Documentation

Release 3.0

The Austerweil Lab at UW-Madison

Sep 12, 2018

Contents

1 What Can Jarjar Do For Me? 3

2 Quickstart 5

3 Detailed Documention 7

Python Module Index 15

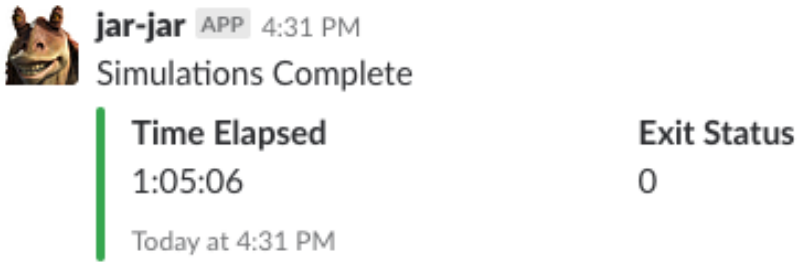
Jarjar is a python utility that makes it easy to send slack notifications to your teams. You can import it as a python module or use our command line tool.

CHAPTER 1

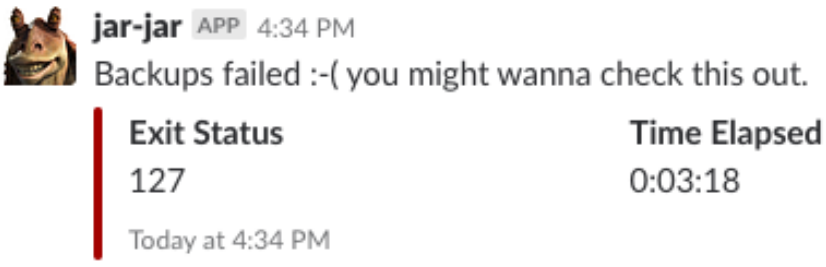
What Can Jarjar Do For Me?

Jarjar was developed at the Austerweil Lab at UW-Madison as a tool for scientists. We use it for all sorts of things, such as:

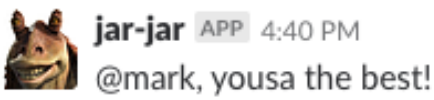
1. Sending a message so that we know when long-running processes have finished.



2. Sending notices when scheduled tasks have failed.



3. Sending out daily positive vibes.



CHAPTER 2

Quickstart

2.1 Install

Installation is simple!

```
pip install jarjar
```

My guess is that you'll want to create jarjar's config file, `.jarjar`. This tells jarjar what you'd like to use as a default for your slack team's webhook, the channel to post to, and the message it sends. Don't worry, you can over-ride these anytime.

Jarjar automatically looks for `.jarjar` in the current working directory as well as the user home (~), so edit this snippet and throw it one of those places:

```
channel='@username'
message='Custom message'
webhook='https://hooks.slack.com/services/your/teams/webhook'
```

If you don't know your team's webhook, you might have to make one.

2.2 Python API

Use the *jarjar python api* like:

```
from jarjar import jarjar

jj = jarjar() # defaults from .jarjar
jj.text('Hi!')

# send an attachment
jj.attach({'meesa': 'jarjar binks'}, message='Hello!')
```

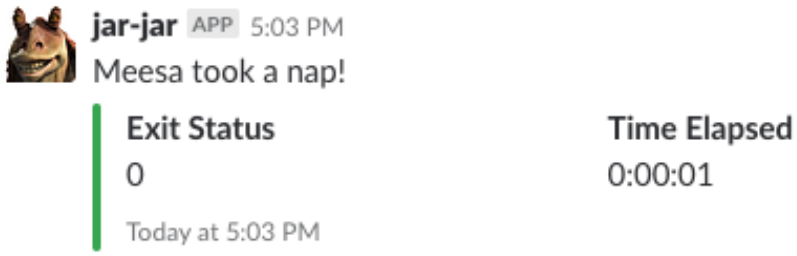
Jajrar also supports *decorator and Jupyter magic workflows*!

2.3 Command Line Tool

We also made a *command line tool* for use outside of python scripts. The command line tool adds functionality to execute processes and send messages when they are complete.

```
jarjar sleep 1 -m 'Meesa took a nap!'
```

And then in your slack team:



Custom attachments are not supported in the CLT at this time, but everything else is:

```
jarjar -m 'Meesa jarjar binks!'
jarjar -m 'Hi, everyone!!' --webhook '<your-url>' -c '#general'
```

CHAPTER 3

Detailed Documention

3.1 Installing jarjar

3.1.1 Just use pip.

We're on pypi.

```
pip install jarjar
```

3.1.2 Config File

You can use jarjar without a config file, but you'll need to tell it your slack webhook and channel each time.

You don't want to live that way.

Jarjar looks to a special config file for a default webhook, channel, and message values. You can over-ride anything in the config file any time but its nice not to have your webhook in each script, amirite??

The file looks like:

```
channel='@username'
message='Custom message'
webhook='https://hooks.slack.com/services/your/teams/webhook'
```

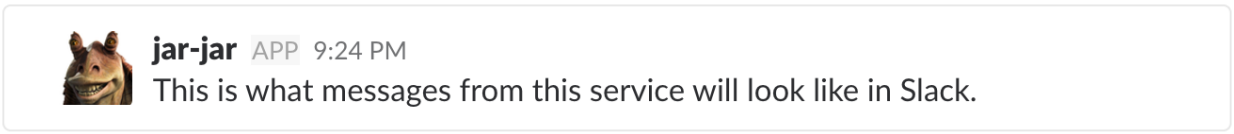
Jarjar looks for values in descending order of priority:

1. Any argument provided to `jarjar().text()` or `jarjar().attach()` at runtime.
2. Any argument provided to `jarjar()` at initialization.
3. Defaults within a file at a user-specified path (`config='...'`), provided to `jarjar()` at initialization.
4. Defaults within a config file `.jarjar`, in the working directory.
5. Defaults within `.jarjar`, located in the user's home directory (~).

3.1.3 Configuring Slack

For this to work in the first place, you need to set up a slack webhook for your team.

While you're doing that, you can also specify a custom name and custom icon. We named our webhook robot `jar-jar`, and we used this icon, so messages look like this:



A note about old vs new-style webhooks

These days slack suggests users configure webhooks through an app, but you can still set up an old-style webhook. Jarjar was written to use the old style-hooks, but both kinds will work - *with one caveat*.

Under the new webhook setup, individual webhooks send messages to a single channel, so Jarjar's `channel='@me'` functionality will not work. Jarjar expects to use an old-style hook so it requires a channel to be specified even if you are using a new-style hook (sorry about that!).

3.2 Using the jarjar command line tool

The CLT provides basic posting functionality like in the python API but it also provides a useful task execution facility.

3.2.1 Posting to your team

The jarjar CLT offers all the functionality of the python API, except for posting attachments (sorry). Posting messages is super easy though!

You can use your defaults from .jarjar

```
jarjar --message 'Meesa jarjar binks!'
```

Or not.

```
jarjar -m 'Hi, everyone!!' --webhook '<your-url>' --channel '#general'
```

3.2.2 Running processes with jarjar

We use jarjar to run a lot of longer processes when we don't want to keep our terminal sessions around. You can use jarjar for this sort of thing.

```
jarjar sleep 3600
```

Generally speaking it is safer to wrap your program in quotes so that its clear which arguments are meant for jarjar and which are meant for your task.

```
jarjar 'python3 simulations.py --niters 100 --out results.csv'
```

Now you can head out for some lunch. Here's what's going on under the hood:

1. **Start up a screen**. The screen can have a custom name (using the `-S` or `--screen_name` flags) but if you don't provide one it'll be named using the program you provide. Above, the screen was named `sleep_3600`.
2. **Run your process in that screen**. If you want you can attach to the screen (using the `-a`, `-r`, or `--attach` flags) and see the magic happen.
3. **Send a message when the process is complete**. If you specified a message (using the `-m` or `--message` flags) jarjar will send it. Jarjar will then kill your screen if:
   - You don't tell it to keep the screen (using the `--no-exit` flag).
   - You didn't attach to it (using the `-a`, `-r`, or `--attach` flags).
   - The program you ran exited with status 0.

Examples

```
# send a custom message
jarjar python run-simulations.py --message 'Simulations Complete!'

# name your screen
jarjar sleep 1 --screen-name 'snooze'

# watch the magic happen
jarjar <program> --attach

# keep the screen around for debugging
jarjar <program> --no-exit

# show jarjar version
jarjar --version

# get help
jarjar --help
```

3.2.3 Argument Reference

- `-h`, `--help`. Show help message.
- `-v`, `--version`. Show jarjar version.
- `-m`, `--message`. Specify message to send. This best done in single-quotes (`jarjar -m 'hi'`) but jarjar rolls with the punches (like `jarjar -m hi`).
- `-w`, `--webhook`. Specify webhook to post to.
- `-c`, `--channel`. Specify channel to post to. Unlike in the python module, only one channel can be supplied at a time. Since `#` is interpreted as a shell comment, you'll want to put this in single quotes (`jarjar -c '#general'`).
- `-a`, `-r`, `--attach`. Attach to the screen once the program has started running.
- `-S`, `--screen_name`. Specify the name of the screen created for the program.
- `--no-exit`. Don't exit the screen even if the program exited successfully.
- `--force-exit`. Exit the screen regardless of your task's exit status.
- `--no-jarjar`. Run a program but don't send a slack message about it. In this case jarjar is just acting as a screen generator.

3.3 API Documentation

class jarjar.**jarjar**(*config=None*, ***defaults*)

A jarjar slack messenger.

This is largely a wrapper around functionality in `requests.post()` with facilities to store and set default values for the desired message to send, channel to post within, and slack team webhook.

Inference for these values proceeds as follows.

1. Any argument provided to *`text()`* or *`attach()`* supersedes all defaults.
2. Any argument provided to this class at initialization.
3. Defaults within a config file at a user-specified path (`config='...'`).
4. Defaults within a config file `.jarjar`, within the working directory.
5. Defaults within `.jarjar`, located in the user's home directory.

The config files (for numbers 3-5) look like:

```
channel="@username"
message="Custom message"
webhook="https://hooks.slack.com/services/your/teams/webhook"
```

If the channel or webhook arguments are never provided, an error is raised. If the channel and webhook are provided but not a message or attachment, jarjar will make something up.

If a value is found in multiple places, the value from the highest priority location is used.

Parameters

config [str, list] Optional. Path(s) to jarjar configuration file(s). If a list, configs should be in descending order of priority.

message [str] Optional. Default message to send.

channel [str, list] Optional. Name of the default channel to post within.

webhook [str] Optional. Webhook URL for the default slack team.

Methods

attach(attach, channel=None, webhook=None, message=None)	Send an attachment. User may also include a text message.
text(message, channel=None, webhook=None, attach=None)	Send a text message. User may also include an attachment.
set_webhook(webhook)	Set jarjar's default webhook.
set_channel(channel)	Set jarjar's default channel.
set_message(message)	Set jarjar's default message.
decorate(func=None, **jj_kwargs)	Decorate a function to send a message after execution.
register_magic(name='jarjar', quiet=False, **kwargs)	Register a magic for Jupyter notebooks.

attach(*attach=None*, ***kwargs*)

Send an attachment.

This method is largely identical to *text()*, only differing in the first argument (attach), which is expected to be a dictionary.

Parameters

attach [dict] Attachment data. Optional *but weird if you don't provide one*. All values are converted to string for the slack payload so don't sweat it.

message [str] Text to send. Optional. If attach is None and there is no default *and* you don't provide one here, jarjar just wings it.

channel [str, list] Optional. Name of the channel to post within. Can also be a list of channel names; jarjar will post to each.

webhook [str] Optional. Webhook URL for the slack team.

Returns

response [requests.models.Response] Requests response object for the POST request to slack.

decorate (*func=None*, ***jj_kwargs*)

Decorate a function to send a message after execution.

This is a simple decorator to compute elapsed time and catch exceptions within a function execution. You can set the usual jarjar kwargs within the decorator. Decorate your function like:

```
jj = jarjar(channel='...')
@jj.decorate
def simulate(x):
    # ...

@jj.decorate(message='...')
def simulate(x):
    # ...
```

Parameters

****jj_kwargs** [keyword arguments] Arguments passed to *attach()*.

register_magic (*name='jarjar'*, *quiet=False*, ***kwargs*)

Register a jarjar Jupyter cell magic.

This magic sends a message whenever its cell executes. The message includes attachments for elapsed time and shows the exception trace if there was one.

Use it like:

```
jj = jarjar(channel='...')
jj.register_magic(message='Cell executed!')

# %% --- new cell ---
%%jarjar
# ... do some stuff! ...
```

Parameters

name [str] Name of the magic to register. Default: 'jarjar'.

quiet [boolean] Flag indicating whether to print the name of the magic.

****kwargs** [keyword arguments] Arguments passed to *attach()*.

set_channel(*channel*)
Set default channel.

Parameters

channel [str] Name of the channel to post within.

set_message(*message*)
Set default message.

Parameters

message [str] Default message to send.

set_webhook(*webhook*)
Set default webhook.

Parameters

webhook [str] Webhook URL for the slack team.

text(*message=None*, ***kwargs*)
Send a text message.

This method is largely identical to *attach()*, only differing in the first argument (`message`), which is expected to be a string.

Parameters

message [str] Text to send. Optional *but weird if you don't provide one*. If attach is None and there is no default *and* you don't provide one here, jarjar just wings it.

attach [dict] Attachment data. Optional. All values are converted to string for the slack payload so don't sweat it.

channel [str, list] Optional. Name of the channel to post within. Can also be a list of channel names; jarjar will post to each.

webhook [str] Optional. Webhook URL for the slack team.

Returns

response [requests.models.Response] Requests response object for the POST request to slack.

3.4 Python API Workflows

Jarjar is great for letting you know when come snippet of code has finished executing, but configuring things properly can be a little bit of a hassle.

A common workflow involves writing your code and then throwing a jarjar call at the end:

```
from jarjar import jarjar
jj = jarjar()

def fun(long_list):
  results = []
  for i in long_list:
    if i == 'something':
      results.append('something')
    else:
```

(continues on next page)

(continued from previous page)

```
      results.append('something else')
  return results

# run the process, notify on completion
results = fun(a_long_list)
jj.text('Process complete!')
```

That looks good, but what if an exception was raised on the way? So maybe you edit like so:

```
try:
  results = fun(my_long_list)
  jj.text('Process complete!')
except Exception:
  jj.text('Process Failed?')
```

That's great. But what if you want to run many such processes? Or what if you want to run the same process twice, getting a notification each time? What if you wanted to include the traceback within the message if there was an exception?

You'll end up writing a lot more code just to handle jarjar notifications. Luckily, we wrote that code for you.

3.4.1 Jarjar decorator

You can decorate a function and jarjar will handle exceptions for you. You'll get a notification whenever the function exits, and it will include the traceback if there was an exception:

```
@jj.decorate
def fun(long_list):
  # ...

results = fun(my_long_list)
```

3.4.2 Jupyter cell magic

Jupyter notebooks are becoming a standard in scientific work. Packaged with jarjar is a Jupyter magic so that users can be notified about a cell's execution.

You first need to register the magic, and then you can use it freely. In one cell:

```
from jarjar import jarjar
jj = jarjar()

jj.register_magic()
```

Then in a later cell:

```
%%jarjar
results = fun(my_long_list)
```

You'll get a notification whether your cell executed successfully or not, and it will include the traceback if there was an exception.

Python Module Index

j

jarjar, 10

Index

A

attach() (jarjar.jarjar method), 10

D

decorate() (jarjar.jarjar method), 11

J

jarjar (class in jarjar), 10
jarjar (module), 10

R

register_magic() (jarjar.jarjar method), 11

S

set_channel() (jarjar.jarjar method), 12
set_message() (jarjar.jarjar method), 12
set_webhook() (jarjar.jarjar method), 12

T

text() (jarjar.jarjar method), 12